Attorney Docket Number PD-02W207 *PATENT*

Defining Breakpoints and Viewpoints in the Software Development Environment for Programmable Stream Computers

Thomas Robert Woodall

Defining Breakpoints and Viewpoints in the Software Development Environment for Programmable Stream Computers

Background of the Invention

This application is a continuation in part of U.S. Patent and Trademark Office application S/N 10,052,082 titled "Reconfigurable Processor Architectures".

Field of invention

This invention is in the field of software development using breakpoints and viewpoints for use in stream computers.

Description of the Related Art

Software, the instructions for a computer to follow, are necessary to perform a particular function and render the computer useful. Developing a properly executing software package, possibly comprising thousands of computer instructions, or steps, is an arduous task. Ideally, in a quality software package, all computer instructions are correct and in the proper sequence, errors have been identified and corrected, and desired functions completely implemented.

In arriving at a quality software package, generally an iteration method is used wherein executable instructions, or code, are written into registers or memory locations and then presented for execution on the target computer. After examining results of code execution, its effects on the registers and memory locations, the code is modified to correct any errors that may have been found. Thus, during the software development process, code compatible tools are needed that allow the examination of various machine states at certain points in the execution of a program to insure that the software is performing the desired function and specified results are reached. Like the oscilloscope is a tool allowing the builder of electronic circuits to "see" circuits in operation, a similar tool is developed for the software engineer. This tool allows

the examination of registers and memory locations following the particular execution of one or more instructions thus tracing the changes brought on by executing the computer instructions.

In prior art, Von Newman type, central control computers, progress of execution of computer instructions is typically performed by examining register contents at certain time intervals corresponding to events chosen by the programmer. These events are defined with the aid of a a compiler or other software tool having the capability of stopping execution and presenting the specific contents of registers and memory locations for inspection and evaluation.

Prior art software tools allowing the step by step examination of registers and memory location in Von Newman architecture computers are well known. However, with the advent of distributed, stream computers described in the parent application, the software tools relevant in single central processing unit (CPU) Von Newman architectures are no longer applicable, in fact, impossible to use. The prior art tools depend on a central entity, the central processing unit to transfer and process information between various parts of a computer. Thus, this one central entity, is responsible for program progress and is the area to be monitored. In this invention, in contrast, there are pluralities of independent processing entities, having no relationship to the Von Newman structure, function or concept.

Summary of the invention

Above limitations are solved by the present invention by a stream computer, said stream computer comprising a first plurality of interconnected functional units. The functional units are responsive to a data stream containing data and tokens. The data is to be operated on by one or more of said first plurality of interconnected functional units.

A second plurality of said interconnected functional units, also part of said stream computer, are allocated for comparing said data stream with a debug stream to generate debug signals. Reporting logic is responsive to the debug signals for

reporting the occurrence of matches between said data stream and said debug stream in a format compatible for human perception.

The second plurality of interconnected functional units generate viewpoints and breakpoints in response to similarities between the data streams present within said stream computer and the debug stream(s) . The breakpoint can either interrupt the data stream or pass it through. Viewpoints do not change the streams they examine, but only extract information therefrom matching said debug stream.

The reporting logic is compatible with a graphical user interface, where the graphical user interface identifies the functional units, inputs and outputs (streams) to or from the functional units, as well as streams between the functional units.

Brief Description of the Drawing

In the Drawing:

FIG. 1 is an exemplary configuration of prior art Von Newman, single control computer architecture.

FIG. 2 is an exemplary graphic representation of a problem to be solved using stream computers;

FIG. 3 is an exemplary user defined function of the problem in fig 2 via a graphical interface;

Fig 4 is the user defined function of fig 3, further refined and ready to run on a target computer or compatible simulator;

Fig 5 shows a typical breakpoints and viewpoints depiction for the Graphical Programming Environment of this invention for the problem of fig 4;

Fig 6 shows details of breakpoint and viewpoints applied to the exemplary

stream computer described herein;

Fig 7 shows an example of digital logic implementing a breakpoint applicable to this invention;

Fig 8 shows an example of digital logic implementing a viewpoint applicable to this invention;

Fig 9 shows an example of allocating part of a stream computer to implement the digital logic required to generate breakpoints and viewpoints.

Detailed Description

The present invention describes an apparatus and method of inducing breakpoints and initiating viewpoints during the programming of a stream computer.

The stream computer comprises a first plurality of interconnected functional units. The functional units are responsive to a data stream containing data and tokens. The data is to be operated on by one or more of said first plurality of interconnected functional units.

A second plurality of said interconnected functional units, also part of said stream computer, are allocated for comparing said data stream, as well as other streams internal to the computer, with a debug stream to generate debug signals. Reporting logic is responsive to the debug signals for reporting the occurrence of matches between said data stream and said debug stream in a format compatible for human perception.

The second plurality of interconnected functional units generate viewpoints and breakpoints in response to similarities between the data streams present within said stream computer and the debug stream(s) . The breakpoint can either interrupts the data stream or pass it through. Viewpoints do not change the streams they examine.

The reporting logic is compatible with a graphical user interface, where the graphical user interface has symbols identifying in human compatible format the functional units, and inputs and outputs (streams) to or from the functional units.

Stream computers are collections of functional units (nodes) operationally interconnected via software streams in addition to hardware links. More specifically, stream computers comprise a plurality of similar functional units where the specification of the computer (and the programming thereof) is defined by the operations performed by the functional units and the streams (also known as data flows) that connect, or traverse the functional units. Thus, the specification of a computer program for stream computers is defined by the operations performed by the functional units and the streams (also known as data flows) that connect the functional units.

The functional units perform prescribed operations on the input(s) and create output(s) when all inputs are valid and the output flow can accept the output. The software streams flowing between functional units contain data, for example, integer data. Some stream computers also have control/token information as part of the data, generally at most a few bits, or a small percentage of the data bits. This control/token information also flows along with the data between functional units. The token information is not operated on (e.g. added, multiplied) but rather is used to determine what operation the functional units perform.

In contrast to stream computers, fig 1 is an exemplary Von Newman computer structure of the prior art. A control unit 107 monitors the transfer of digital information between memory 101 , Arithmetic and Logic Unit (ALU) 103, and Input/Output unit 105. Typically, control unit 107 will instruct ALU 103 to fetch contents of a particular location within memory 101, perform a mathematical or logical operation on the retrieved data, then write the results at another location within memory 101. Control unit 107 can interpret an instruction retrieved from memory 101and select alternative courses of action based on the results of previous operations. Memory 101 contains both data and instructions to be followed by control unit 107 and/or ALU 103. The instructions can be followed in any order. Similarly, control unit 107 will instruct Input/Output (I/O) unit 105 to make ready data for ALU 103 to read. Thus, for software development purposes, monitoring control unit 107, contents of memory

101 and I/O 105 will suffice to ascertain the state of the computer on a step by step basis and monitor progress of program execution. Compilers of the prior art provide break points within a sequence of executed steps by control unit 107. The break points typically halt the execution of program steps within control unit 107 and list contents of certain locations within memory 101 for inspection and reconciliation with expected values. In Von Newman architectures, there is no flow of data to a plurality of functional units, each capable of computing the required results. Von Newman architectures can be viewed as having centralized control, while stream computers represent a distributed structure with much duplication of individual functional units where each functional unit reacts independently of the others and computes results under control of flowing data and tokens.

To accommodate the distributed nature of stream computers, and facilitate software development for them, this invention inserts breakpoint and viewpoint logic into the definition (or program) of a programmable stream computer. This functionality is best implemented in a graphical programming environment and the examples illustrating the best mode of the invention use such an environment.

The examples in this invention are illustrated by the use of a simple polynomial equation to illustrate the characteristics of this invention. Typically a large number, perhaps hundreds, of functional units exemplified below are interconnected to compute Fast Fourier Transforms, or matched filter functions.

A simple, general illustrative equation is used:

$$Y = PX^2 + QX + R$$

where P, Q, R and X are inputs and Y is the output. For simplicity, X and Y are the only dynamic data streams and P, Q, and R are constants (i.e. registers preloaded with constant values) whereas a new value of X is potentially available on the input stream for every clock pulse.

As shown in fig 2, in this example, the fundamental (primitive) functional unit in the sample computer is a functional unit 202 capable of performing a multiplication in multiplier 206 and an addition in adder 208. The results from multiplier 206 are

combined with an addition in adder 208. While this example shows a multiplier 206 and adder 208, any other functions can be substituted, depending on computational needs.

Functional unit 202 is shown by the dashed line and in later figures will be identified as a MUL. Functional unit 204 is structurally similar to functional unit 202, having a multiplier 212 and adder 214 to generate an output Y.

Delay 210 applied to X matches the delay experienced in functional unit 202. Without delay 210, X would be presented to functional unit 204 before the result from functional unit 202, $PX+Q$, was ready. This facilitates keeping the pipeline full.

From the diagram in Fig 2, the output Y is the result of $PX^2 + QX + R$, as computed by functional units 202 and 204.

Breakpoints and Viewpoints

In a graphical programming environment to be used with this invention, the user graphically represents the function being performed. In this example,as shown in fig 3, a user defined function for the same equation as in fig 2 $Y = PX^2 + QX + R$ is being defined.In the graphical interface, MUL 301 is the hardware functional unit 202, while MUL 303 is the hardware functional unit 303. The delay 305 for X is also provided to facilitate the operation of the pipeline.

In the graphical interface, for this example, the user selects each data flow and selects the data types. The user also selects the function and specifies the function to be performed by MUL 301 and MUL 303. In this case, the function of MUL 301 and MUL 303 is a multiply with an add.

Typically, having completed the specification of the functional units, the specification will be executed on a simulator or directly in hardware. The graphical representation shown in fig 4, is an example of the structure of fig 3. The graphical representation is compiled to generate an executable program. The executable to compute $Y = 2X^2 + 3X + 4$ is then run on a target computer or simulator. The details

of compilation of a specification, and running the compilation on a target (whether real hardware or a simulation) are well known, and therefore not detailed further.

The development environment of this invention inserts the breakpoint and viewpoint specifications into the executable program generated from fig 4. The breakpoint and viewpoint specifications are loaded as part of the program image in fig 5. The capabilities are fully programmable. The graphical programming environment incorporates the details. The programmer sees the view as shown in fig 5. The other mechanistic, underlying details are conveniently shielded from the user to facilitate the development process.

Once a breakpoint and/or viewpoint is defined, the graphical interface saves the definition but allows the user to determine whether or not specific breakpoints and viewpoints are incorporated into the specific program image being created.

The graphical programming environment is tied to the target as shown in fig 4. Fig 4 shows the combination of functions MUL reading the X values from a file, source of X values 402, and deposits the Y outputs into another file, or storage means, store results Y 404. The specification defines the values of P, Q, and R to be 2, 3, and 4 respectively, so that the sample equation now becomes

$$Y = 2X^2 + 3X + 4$$

When the user runs this program the input file in 402 is read and the output file in 404 is written. If the programmer has incorrectly specified the function to be performed by one or both functional units (identified as MUL in the graphic interface) 301 and 303 units then the output Y may not be as expected. That is, there may be an error in the program. With an error present, the programmer inserts a breakpoint in an attempt to locate the error. This invention allows a breakpoint to be inserted on any data stream. Therefore, the user can select the input stream X, the stream flowing between MUL units 301 and 303, or the output stream Y. Since each stream has two components, namely data and token, the user is able to select a combination of the two. For example, a condition for generating a breakpoint is defined by:

a) when the data is greater than zero and

b) the token equals zero.

The capabilities of the stream computer functional units will define the set of available breakpoint conditions. However, the breakpoint function can use one or more functional units. The number of breakpoint functions is only limited by the number of programmable functional units and connectivity defined by the programmable stream computer.

In the example of Fig 5, the programmer introduces a breakpoint triggered by the data stream between MUL 301 and MUL 303 having a value of 1. The programmer also wants to view the X input stream to the second MUL 303 after delay 305. For the example shown, the corresponding value of X is -1. Figure 5 shows how the user specifies this breakpoint graphically using the graphic interface. Breakpoint 501 specifies the data condition and token condition. Data condition for breakpoint 501 is $D : 1$ while token condition is $T : ANY$. The AND function indicates that both conditions must be met for a breakpoint to be triggered.

Viewpoint 503 indicates the condition necessary to generate a viewpoint. For example, the data value 0 is shown, followed by a semicolon, and then the token value 0.

Viewpoint 505 monitors the stream of X, after delay 305, before entering MUL 303. The viewpoint is triggered when the data value is 0 and token value is also 0.

The display of the graphic interface shields the programmer from the underlying details and facilitates the programming process. Figure 6 shows an exemplary implementation for the breakpoints shown in figure 5 using a similar level of abstraction. In fig 6, viewpoints 604, and 606 and breakpoint 602 are interposed to examine the data streams from MUL 301 and into MUL 303.

Fig 7 shows an example of a breakpoint implementation applicable to this invention. The breakpoint uses digital logic and uses an independent stream (the debug stream) to perform comparisons and identify the occurrence of a condition for triggering the breakpoint. In this digital logic, ALU 701 compares a debug stream

and breakpoint number and stores results in a memory 703. The quantities stored in memory 703 are compared by ALU 705 with the data stream to identify a breakpoint.

The structure the digital logic made of ALU 701, memory 703 and ALU 705 used to identify the breakpoint is the same as that used within the functional units. An ALU is the same functionally, and can be the same as a MUL functional unit without the multiply capability, as shown in figs 6 and 7. The debug stream carries information to the digital logic and enables output of the results.

Because the functional units have the same structure required for breakpoint generation, a programmer has the option to allocate part of the stream computer resources for breakpoint generation, on a case by case basis. If breakpoints are specified in the program, they require valuable resources within the stream computer, namely streams and functional units.

In the example shown in fig 7, the input to initiate a breakpoint is a debug stream that will indicate whether the breakpoint is enabled. A combination of data and token can be used. For example, the data may be a unique identifier for each breakpoint so that more than one can be used. The output stream will indicate if the breakpoint has been triggered or else it will pass along the input. The viewpoints then take this information (also passing it along for downstream viewpoints) and act accordingly. If the breakpoint has been triggered, then on the first available clock it may forward breakpoint trigger information.

On the next clock cycle, the digital logic of fig 7 may output the identifier of the viewpoint and then finally the value of the viewpoint to a reporting logic. The token values that are input to and output by the viewpoints are:

null (data is invalid);

breakpoint (data is the address of the breakpoint and enable/disable/trigger data);

viewpoint identifier (data is the viewpoint identifier);

viewpoint value (the data is the value of the viewpoint).

The Breakpoint in fig 7 is implemented by having ALU 701 and 705 compare the debug stream input. If the token indicates breakpoint , ALU 701 will detect if the breakpoint number (a constant) is a match, if so it will output a breakpoint hit. This detection by ALU 701 controls digital logic that will generate output that is the same as this input until a breakpoint is hit. If a breakpoint has been hit then the digital logic will neither consume the input (causing upstream functions to stop), nor produce output causing downstream functions to stop. This information also continues down the debug stream to viewpoints so they will output their data.

The viewpoint in fig 8 is implemented by splitting the data stream and always forwarding it unchanged. The digital logic on the debug stream, viewpoint ALU 802, looks to see if any breakpoint has been detected. If so, the digital logic will insert its viewpoint identifier and data stream values into the debug stream at the first available opportunity and will continue to do so until no data stream information is available or the breakpoint is released. Memory 804 provides the incoming data stream to viewpoint ALU 802 for viewpoint detection in accordance with the debug stream, as well as passing the data stream unchanged to the next stage or functional unit.

These implementations are examples indicative of the concept of the invention. The exact implementation will be a function of the capabilities of the functional units and available stream connectivity in the system. Breakpoints and viewpoints are implemented using the same programmable functional capabilities as the typical, normal function units available within a stream computer. These functions are inserted into the program of the stream computer the same as any other functionality. Because of this and the fact that no special debug capabilities are required and that generally differ in hardware versus a simulation, there will be a consistent implementation in the simulation and the target thereby easing program development.

This invention does not preclude special debug capabilities, but rather frees the software development engineer from relying on them. This differs from the prior art because most processors require and implement special functions to support debug-

ging. Using this paradigm no special capabilities are required, although the system does require some functional units to be available for debug. However, having spare units would be normal during unit testing where small fragments of the overall program are being tested.

The concept is further summarized using the concepts from fig 2 to fig 8 in a simple example shown in Fig 9. Here, functional units 901, 903, 905, 907, 909, 911, 913, 915 and 917 form a stream computer and have access to the data and token stream. Of these, functional units 905 and 911 are allocated to process breakpoint / viewpoint generation. Functional units 905 and 911 form digital logic 919 required to interpret a debug stream. Functional units 905 and 911 are cooperatively associated with, for example, functional units 903 and 909 respectively, in that the data stream through 903 and 909 are monitored by 905 and 911 respectively.

The results from digital logic 919 are transferred to reporting logic 921 for conversion to a format compatible with the display of breakpoints, viewpoints and associated data stream content 923. In effect, functional units 905 and 911, part of the stream computer are programmed to respond to the debug stream and the data/token stream for debugging purposes. It is envisioned that the functional units making up the stream computer as well as reporting logic 921 are integrated on a single semiconductor or hybrid substrate to minimize size and cost.

The structure of the stream computer applicable herein is discussed in the parent application, and is incorporated herein in its entirety.

Although presented in exemplary fashion employing specific embodiments, the disclosed structures are not intended to be so limited. For example, although a multiply and add function is shown as a building block example, any other combination of mathematical or Boolean logic operation could benefit from the concepts described by the invention herein.

Those skilled in the art will also appreciate that numerous changes and modifications could be made to the embodiment described herein without departing in any way from the invention. These changes and modifications and all obvious variations

of the disclosed embodiment are intended to be embraced by the claims to the limits set by law.